Programación 4

EXAMEN DICIEMBRE 2015

Por favor siga las siguientes indicaciones:

- Escriba con lápiz y de un solo lado de las hojas.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Recuerde entregar su número de examen junto al examen.
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el parcial.

Problema 1 (35 puntos)

Parte a)

1) Defina el concepto tipo asociativo.

2) ¿Qué es un DSS, y para qué se utiliza?

Parte b)

Se desea construir un sistema de gestión para una librería virtual. La librería vende tanto libros en formato papel como en formato electrónico (ebook).

De los libros (en ambos formatos) interesa conocer su conjunto de autores, nombre del editor, precio e ISBN (supondremos que este dato es una cadena de caracteres de tamaño 10, formado por tres letras al principio y el resto de caracteres son numéricos). El precio de un libro en cualquiera de los dos formatos debe ser el mismo. Los libros se identifican por su conjunto de autores y el título. De los libros en formato papel, interesa conocer la cantidad que hay en stock.

Solamente los usuarios registrados al sistema pueden realizar compras, de estos interesa conocer su nombre, apellido, email (que lo identifica).

De los autores interesa conocer su nombre, apellido y país de nacimiento. Cada uno de ellos es identificado por un valor numérico denominado id Creador, y conjunto de libros de los cuales es autor o coautor.

Se desea mantener un registro de todas las ventas de libros realizadas, de estas interesa conocer el usuario comprador, el libro vendido, precio y la fecha de la transacción. Los precios de los libros pueden variar con el tiempo. Tenga en cuenta que un usuario puede comprar el mismo libro más de una vez, tanto en formato digital (ebook) como papel, y en cada compra solo puede adquirir una unidad.

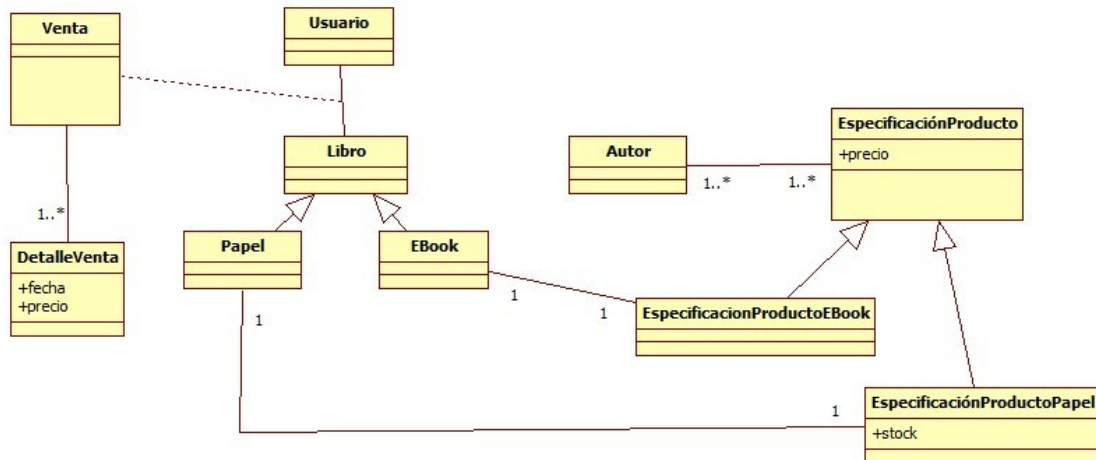
Considere el siguiente Caso de Uso:

Caso de Uso:	Consultar ventas de un libro
Actores:	Administrador
Descripción:	El caso de uso comienza cuando el usuario administrador desea consultar información sobre las ventas de determinado título. Para esto el usuario ingresa el título del libro, y su conjunto de autores, para el libro que desea consultar. En caso de que el libro ingresado no existir en el sistema, el caso de uso finaliza. De lo contrario, el sistema pide al usuario que indique el tipo de ventas que quiere visualizar (ebook , papel o ambas), y luego despliega en pantalla la información que interesa conocer: Email de usuario comprador, fecha de venta, título del libro, y precio en el cual fue vendido.

i) Se pide: Modelo de Dominio, con restricciones en lenguaje natural.

ii) Se pide: DSS para el caso de uso “Consultar ventas de un libro”.

Solución:



Restricciones:

-ISBN es una cadena de caracteres de tamaño 10, formado por tres letras al principio y el resto de caracteres son numéricos.

-El email identifica a los usuarios

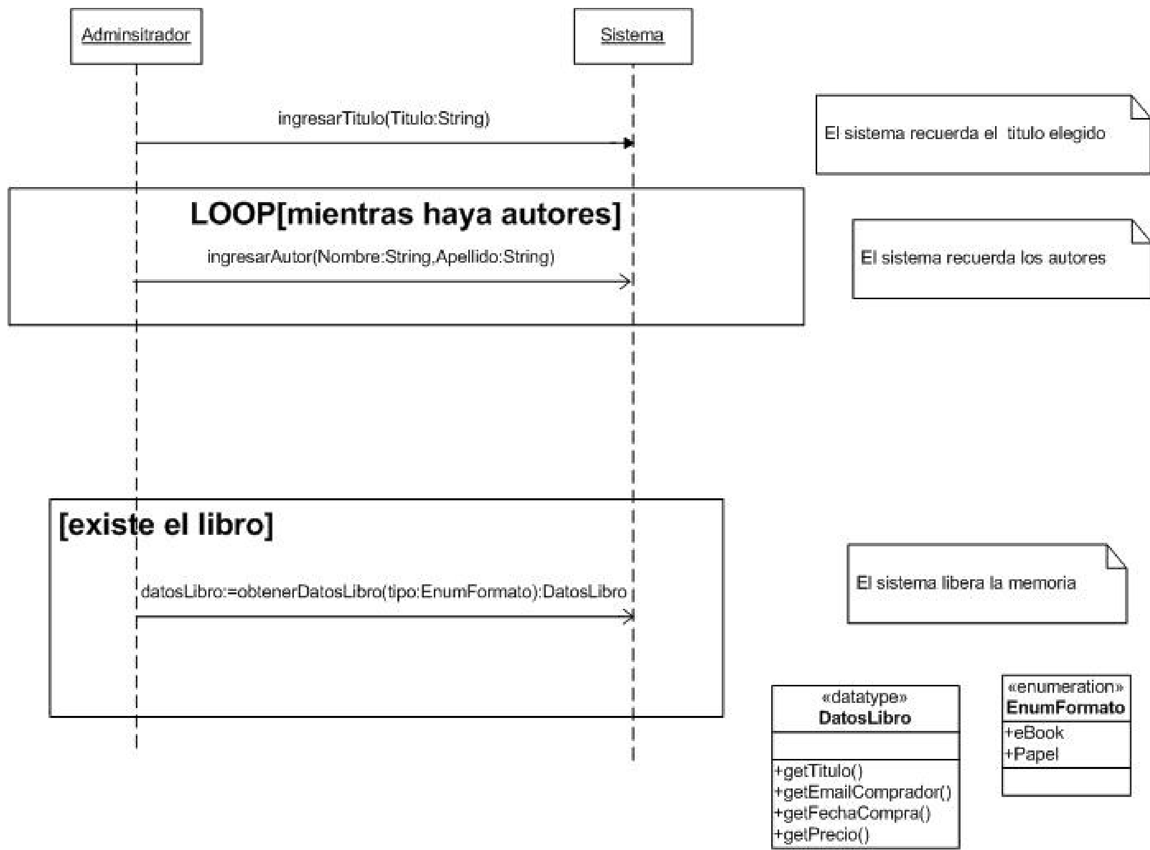
-El idcreador identifica a los autores

-Los libros se identifican por el conjunto de autores y el título.

-El precio de una venta está dado por el precio actual del libro.

-El stock de determinado libro es mayor o igual a cero

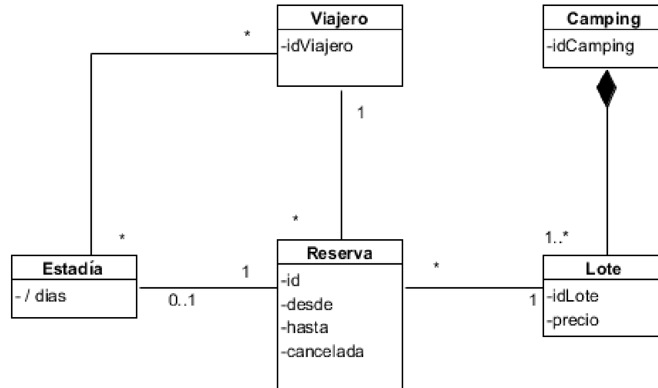
-Dos libros con el mismo conjunto de autores y el mismo titulo, de distinto formato, tienen el mismo precio.


Adminsitrador
Sistema
ingresarTitulo(Titulo:String)
El sistema recuerda el titulo elegido
LOOP[mientras haya autores]
ingresarAutor(Nombre:String,Apellido:String)
El sistema recuerda los autores
[existe el libro]
datosLibro:=obtenerDatosLibro(tipo:EnumFormato):DatosLibro
El sistema libera la memoria
«datatype»
DatosLibro
+getTitulo()
+getEmailComprador()
+getFechaCompra()
+getPrecio()
«enumeration»
EnumFormato
+eBook
+Papel

Problema 2 (35 puntos)

Parte a)

Su equipo de Analistas realizó el siguiente modelo de dominio para soportar un sistema de gestión de reservas de campings:



Restricciones:

- idViajero, idCamping e idLote son únicos para cada instancia de clase.
- El viajero que realiza la reserva no es parte de los otros ocupantes de la estadía.
- Un lote no puede estar asociado a dos reservas cuyos períodos se superpongan.
- La reserva podrá estar cancelada sólo si no tiene una estadía asociada.

Además se generaron los siguientes contratos, que corresponden a operaciones del sistema necesarias para realizar algunos de los casos de uso relevados:

altaReserva (idCamping, idViajero, desde, hasta, idLote: String): bool	
Descripción	Crea la reserva para el viajero en el camping y lote especificado por rango de fecha definido.
Parámetros	- **idCamping**: identificador de Camping - **idViajero**: identificador de Viajero - **desde**: fecha de inicio de la reserva. - **hasta**: fecha de finalización de la reserva - **idLote**: identificador del lote dentro del Camping
Pre-condiciones	- Existe en el Sistema un Viajero con *idViajero*. - Existe en el Sistema un Camping con *idCamping*. - *desde* < *hasta*, ambas fechas válidas posteriores a la actual. - Existe *idLote* dentro del Camping.
Post-condiciones	- En caso de no existir otra reserva para el lote durante el rango de fechas definido, el Sistema genera una nueva instancia de Reserva y Estadía asociadas al viajero y lote correspondientes retornando *true*. - En caso contrario el sistema queda sin cambios y retorna *false*.

cancelarReserva (idViajero, idCamping, idLote idReserva, desde: String)	
Descripción	Cancela la reserva del viajero para el lote indicado con fecha de comienzo *desde*.
Parámetros	- **idCamping**: identificador de camping - **idViajero**: identificador de viajero

	- **idLote**: identificador del lote dentro del camping - **idReserva**: identifica la reserva dentro del lote.
Pre-condiciones	- Existe en el Sistema un viajero con *ciViajero*. - Existe un Lote con *idLote* en el camping *idCamping*. - Existe una reserva para *idViajero* y *idLote* en la fecha *desde*.
Post-condiciones	- Se elimina la instancia de Estadía asociada a la reserva. - Se actualiza el valor del atributo cancelada en la reserva.

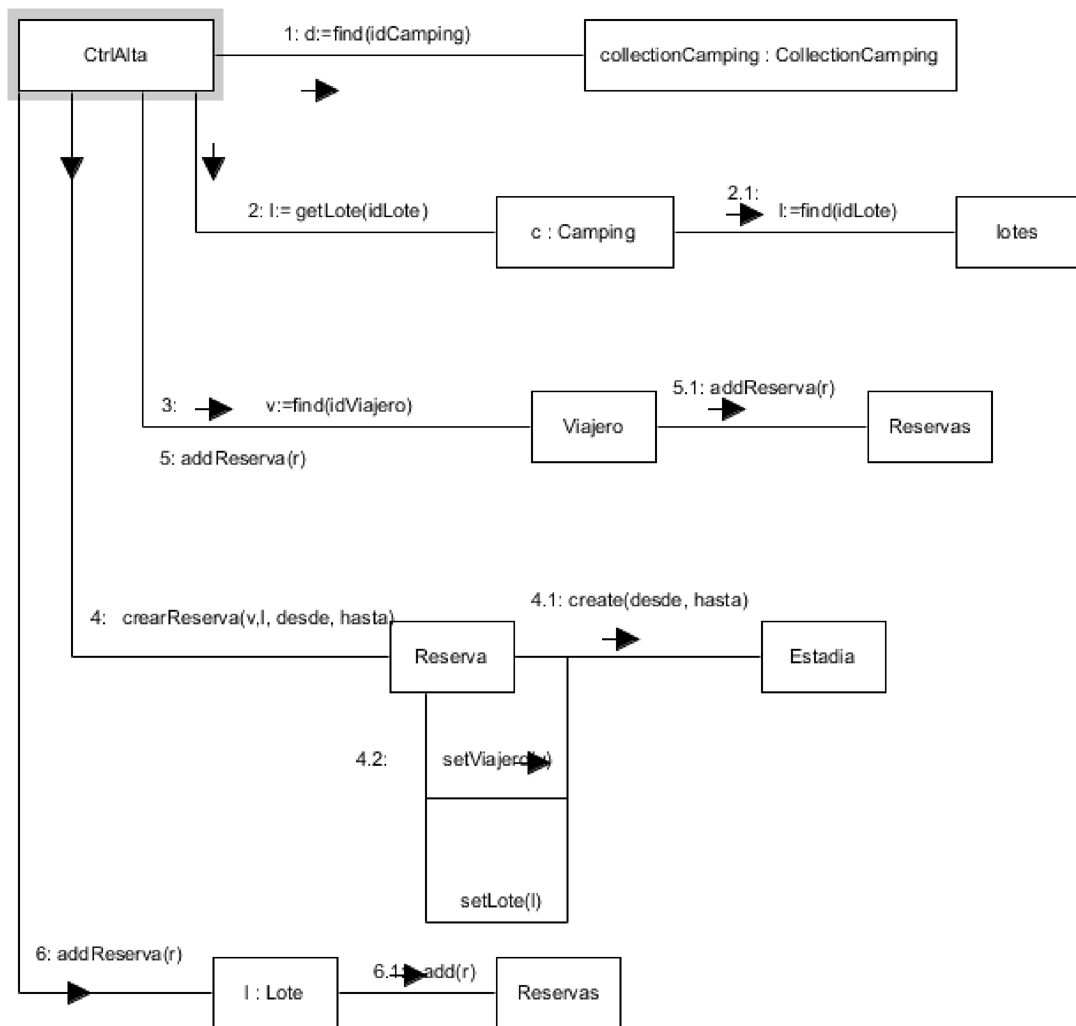
Los viajeros podrán (opcionalmente) suscribirse en los campings que deseen indicando un rango de fechas. De este modo cada vez que se cancele una reserva del camping al que están suscritos y entre las fechas indicadas, serán notificados mediante la operación static public Notificación::notificar(String mensaje).

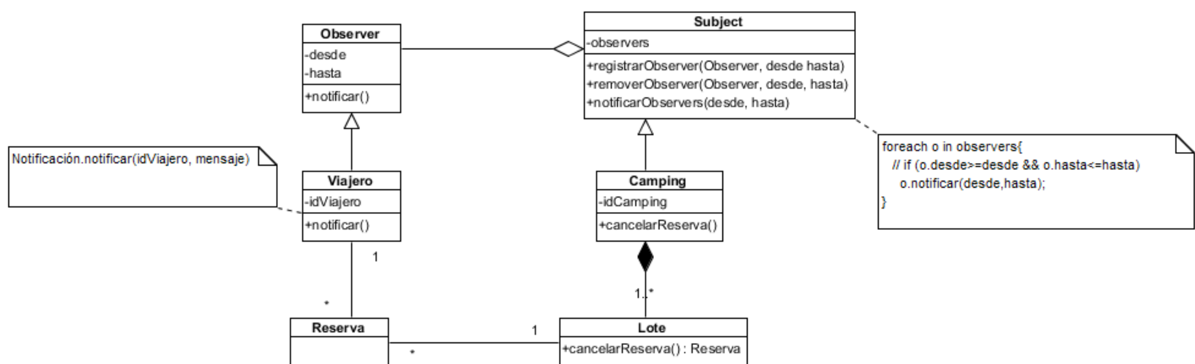
Se pide:

i. Proponga un Diagrama de Clases de Diseño que soporte el mecanismo de notificación de cancelación de reservas. Identifique el/los patrones utilizados.

ii. Realizar el Diagrama de Comunicación (incluyendo visibilidades) para cada una de las operaciones previamente declaradas considerando i.

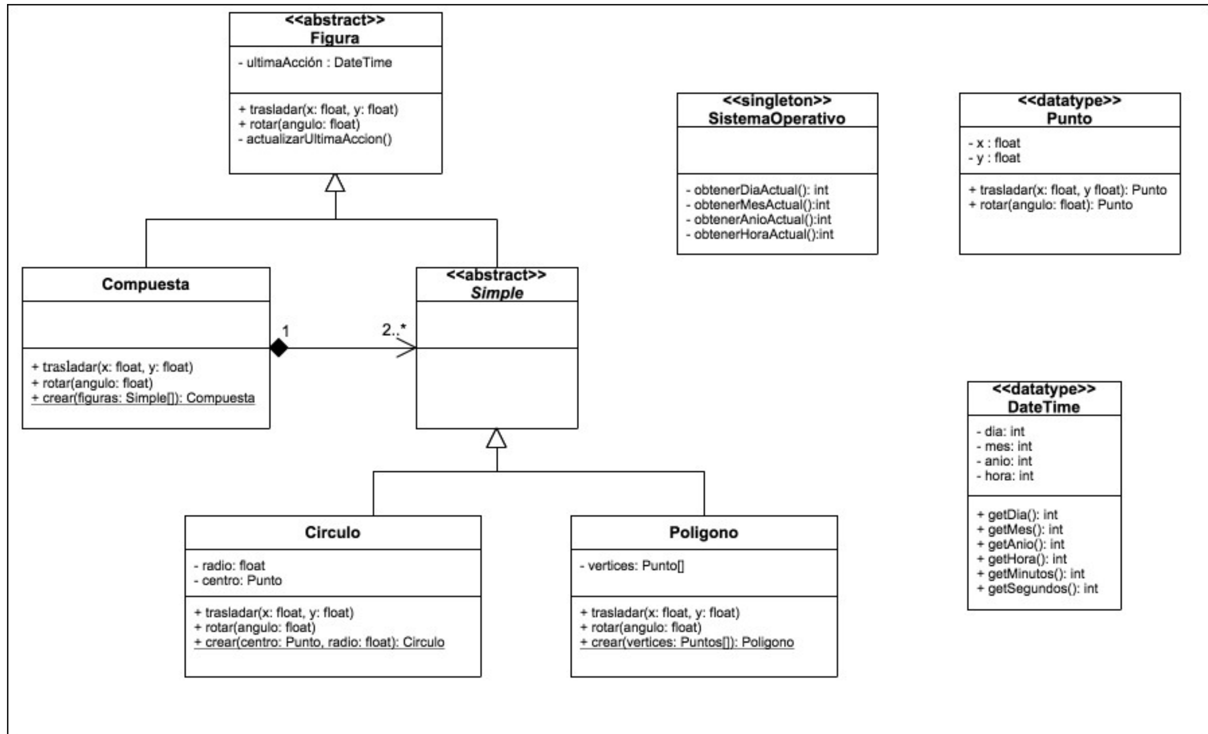
Solución:


CtrlAlta
1: d:=find(idCamping)
collectionCamping : CollectionCamping
2: l:= getLote(idLote)
c : Camping
2.1:
l:=find(idLote)
lotes
3:
v:=find(idViajero)
5: addReserva(r)
Viajero
5.1: addReserva(r)
Reservas
4: crearReserva(v,l, desde, hasta)
Reserva
4.1: create(desde, hasta)
Estadia
4.2:
setViajero(v)
setLote(l)
6: addReserva(r)
l : Lote
6.1: add(r)
Reservas


Observer
-desde
-hasta
+notificar()
Subject
-observers
+registrarObserver(Observer, desde hasta)
+removerObserver(Observer, desde, hasta)
+notificarObservers(desde, hasta)
Notificación.notificar(idViajero, mensaje)
Viajero
-idViajero
+notificar()
Camping
-idCamping
+cancelarReserva()
foreach o in observers{
// if (o.desde>=desde && o.hasta<=hasta)
o.notificar(desde,hasta);
}
1
*
1..*
Reserva
1
*
Lote
+cancelarReserva() : Reserva

Problema 3 (30 puntos)

Se desea implementar un sistema similar a Paint y para esto se le encomendó a usted realizar la implementación de la capa lógica. El sistema maneja *Figuras* que pueden ser *Simples* como *Polígonos* o *Círculos* y *Compuestas* que son conjuntos de al menos dos *Simples*. Inspirados en el patrón *Composite*, el equipo de diseño llegó al siguiente diagrama de clases de diseño:



Nota: Se omiten dependencias

Sobre las figuras se pueden realizar acciones como traslaciones y rotaciones. Cuando se realiza una acción sobre una figura *Simple*, se debe tener en cuenta que en el caso de ser un *Círculo*, dicha acción se aplica en el centro, y en el caso de ser un *Polígono* se aplica a todos sus vértices. Al aplicar una acción sobre una figura *Compuesta* se debe aplicar la acción sobre el conjunto de figuras *Simples* asociadas. También se desea registrar la fecha de la última acción realizada en el atributo *últimaAcción* de la clase *Figura*, que se inicializará con la fecha de creación del objeto. Para esto se cuenta con la clase singleton *SistemaOperativo* que nos brinda el *día*, *mes*, *año* y *hora*. Sin embargo, la hora es un entero que representa la cantidad de segundos transcurridos en el día, por ejemplo si la operación retorna 150090 la hora será 30 segundos (150090 mod 60), 11 minutos ((150090 / 60) mod 60) y 4 horas (150090 / 60 / 60). Se desea almacenar la hora como un entero y utilizar los getters de DateTime para obtener los segundos, minutos y la hora. El datatype DateTime tendrá un constructor por defecto que inicializa un DateTime con el tiempo actual. También recurriendo a buenas prácticas de programación, buscando encapsular y no repetir código, se quiere que la actualización de *ultimaAcción* para cada figura se realice utilizando la operación privada *actualizarUltimaAccion* en Figura que deberá ser llamada por trasladar y rotar en esta clase. Es importante asegurar que cuando se realice la operación trasladar o rotar en un *Círculo*, *Polígono* o *Compuesta* se ejecute de alguna forma la operación privada *actualizarUltimaAccion* de *Figura* de forma de actualizar la fecha. Por último, solamente se podrán crear figuras *Compuestas*, *Círculo* y *Polígono* utilizando las operaciones de clase crear. Esta operación debe chequear que una figura *Compuesta* esté conformada por más de dos figuras *Simple*, que el radio sea mayor que 0 para una figura *Círculo* y que una figura *Polígono* esté conformada por más de dos puntos. En caso de no cumplirse se deberá lanzar la excepción `std::invalid_argument`.

Considerar:

1. Puede suponer la existencia de la interface ICollectible e implementaciones de ICollection (clase List) e IIterator según sea necesario.
2. Es posible utilizar las clases set<T> o vector<T> de la STL.
3. Las implementaciones **deben** incluir constructores y destructores.
4. Implementar los getters **solamente** para datatypes.
5. **No** implementar setters.
6. **No** incluir directivas al precompilador.

Se pide:

1. Implementar .h y .cpp del datatype DateTime.
2. Implementar .h y .cpp de la clase Figura.
3. Implementar .h y .cpp de la clase Compuesta.
4. Implementar .h y .cpp de la clase Circulo.

Solución:

/ SistemaOperativo.h ***/***

```
class SistemaOperativo {

public:

    static SistemaOperativo* getInstance();



    int obtenerDiaActual();

    int obtenerMesActual();

    int obtenerAnioActual();

    int obtenerHoraActual();



    virtual ~SistemaOperativo();

private:

    SistemaOperativo();

    static SistemaOperativo* instance;

};
```

/ SistemaOperativo.cpp ***/***

```
SistemaOperativo* SistemaOperativo::instance = NULL;


SistemaOperativo::SistemaOperativo() {
}


SistemaOperativo* SistemaOperativo::getInstance(){
    if (!instance) {
        instance = new SistemaOperativo();
    }

    return instance;
}


int SistemaOperativo::obtenerDiaActual() {
    return 11;
}


int SistemaOperativo::obtenerMesActual() {
    return 11;
}


int SistemaOperativo::obtenerAnioActual() {
```

```
    return 2015;
}


int SistemaOperativo::obtenerHoraActual() {
    return 150090;
}


SistemaOperativo::~SistemaOperativo() {
}
```

/ Simple.h ***/***

```
class Simple : public Figura {
public:
    Simple();
    virtual ~Simple();
private:


};
```

/ Simple.cpp ***/***

```
Simple::Simple() {
}


Simple::~Simple() {
}
```

/ Punto.h ***/***

```
class Punto {
public:
    Punto();
    Punto(float x, float y);

    Punto trasladar(float x, float y);
    Punto rotar(float angulo);

    virtual ~Punto();
private:
    float x;
    float y;

};
```

/ Punto.cpp ***/***

```
Punto::Punto() {
    this->x = 0;
    this->y = 0;
}

Punto::Punto(float x, float y) {
    this->x = x;
```

```
    this->y = y;
}


Punto Punto::trasladar(float x, float y){
    return Punto(this->x + x, this->y + y);
}


Punto Punto::rotar(float angulo){
    return Punto(this->x, this->y);
}


Punto::~Punto() {
}
```

/ Poligono.h ***/***

```
class Poligono : public Simple {
public:
    std::vector<Punto> getVertices();
    void setVertices(const std::vector<Punto>&);

    static Poligono* crear(const std::vector<Punto>&);

    void trasladar(float x, float y);
    void rotar(float angulo);
```

```
    virtual ~Poligono();

private:

    std::vector<Punto> vertices;

    Poligono(std::vector<Punto> vertices);

};
```

/ Poligono.cpp ***/***

```
Poligono::Poligono(std::vector<Punto> vertices) {

    this->vertices = vertices;

}



std::vector<Punto> Poligono::getVertices() {

    return this->vertices;

}



void Poligono::setVertices(const std::vector<Punto>& vertices) {

    this->vertices = vertices;

}



Poligono* Poligono::crear(const std::vector<Punto>& vertices) {

    if (vertices.size() < 2) {

            throw std::invalid_argument("Los poligonos deben tener mas de 2 vertices.");

    }
```

```
    return new Poligono(vertices);
}

void Poligono::trasladar(float x, float y) {
    std::vector<Punto> newVertices;
    for(std::vector<Punto>::iterator it = vertices.begin(); it != vertices.end();
++it) {
        Punto punto = (*it);
        newVertices.push_back(punto.trasladar(x, y));
    }

    this->vertices = newVertices;

    Figura::trasladar(x, y);
}

void Poligono::rotar(float angulo) {
    std::vector<Punto> newVertices;
    for(std::vector<Punto>::iterator it = vertices.begin(); it != vertices.end();
++it) {
        Punto punto = (*it);
        newVertices.push_back(punto.rotar(angulo));
    }
```

```
    this->vertices = newVertices;

    Figura::rotar(angulo);

}


Poligono::~Poligono() {

}
```

/ Figura.h ***/***

```
class Figura {

public:

    Figura();


    DateTime getUltimaAccion();

    void setUltimaAccion(const DateTime&);


    virtual void trasladar(float x, float y);

    virtual void rotar(float angulo);


    virtual ~Figura();

private:

    DateTime ultimaAccion;

    void actualizarUltimaAccion();

};
```

/ Figura.cpp ***/***

```
Figura::Figura() {

    this->ultimaAccion = DateTime(); //Se puede no poner esta linea inicializa
con el por defecto

}



DateTime Figura::getUltimaAccion() {

    return this->ultimaAccion;

}



void Figura::setUltimaAccion(const DateTime& ultimaActualizacion) {

    this->ultimaAccion = ultimaActualizacion;

}



void Figura::trasladar(float x, float y) {

    actualizarUltimaAccion();

}



void Figura::rotar(float angulo) {

    actualizarUltimaAccion();

}



void Figura::actualizarUltimaAccion() {

    this->ultimaAccion = DateTime();

}
```

```
Figura::~Figura() {

}
```

/ DateTime.h ***/***

```
class DateTime {
public:
    DateTime();

    int getAnio();
    int getMes();
    int getDia();
    int getHora();
    int getMinutos();
    int getSegundos();

    virtual ~DateTime();
private:
    int anio;
    int mes;
    int dia;
    int hora;
};
```

/ DateTime.cpp ***/***

```
DateTime::DateTime() {
```

```
    SistemaOperativo* so = SistemaOperativo::getInstance();
    this->anio = so->obtenerAnioActual();
    this->mes = so->obtenerMesActual();
    this->dia = so->obtenerDiaActual();
    this->hora = so->obtenerHoraActual();
}

int DateTime::getAnio() {
    return this->anio;
}

int DateTime::getMes() {
    return this->mes;
}

int DateTime::getDia() {
    return this->dia;
}

int DateTime::getHora() {
    return this->hora / 3600;
}
```

```
int DateTime::getMinutos() {
    return (this->hora / 60) % 60;
}


int DateTime::getSegundos() {
    return this->hora % 60;
}


DateTime::~DateTime() {
}
```

/ Compuesta.h ***/***

```
class Compuesta : public Figura {
public:
    std::vector<Simple*> getSimple();
    void setSimple(const std::vector<Simple*> simple);


    static Compuesta* crear(std::vector<Simple*> simple);


    void trasladar(float x, float y);
    void rotar(float angulo);


    virtual ~Compuesta();
private:
```

```
    std::vector<Simple*> simples;

    Compuesta(const std::vector<Simple*>& simples);

};
```

/ Compuesta.cpp ***/***

```
Compuesta::Compuesta(const std::vector<Simple*>& simples) {

    this->simples = simples;

}



std::vector<Simple*> Compuesta::getSimple() {

    return this->simples;

}



void Compuesta::setSimple(const std::vector<Simple*> simples) {

    this->simples = simples;

}



Compuesta* Compuesta::crear(std::vector<Simple*> simples) {

    if (simples.size() < 2) {

            throw std::invalid_argument("las figuras compuestas deben ser
formadas por mas de una simple.");

    }



    return new Compuesta(simples);

}
```

```
void Compuesta::trasladar(float x, float y) {

        for(std::vector<Simple*>::iterator it = simples.begin(); it !=
simples.end(); ++it) {

        Simple* simple = (*it);

        simple->trasladar(x, y);

    }



    Figura::trasladar(x, y);

}



void Compuesta::rotar(float angulo) {

        for(std::vector<Simple*>::iterator it = simples.begin(); it !=
simples.end(); ++it) {

        Simple* simple = (*it);

        simple->rotar(angulo);

    }



    Figura::rotar(angulo);

}



Compuesta::~Compuesta() {

        for(std::vector<Simple*>::iterator it = simples.begin(); it !=
simples.end(); ++it) {

        Simple* simple = (*it);
```

```
        delete simple;
    }
}
```

/ Circulo.h ***/***

```
class Circulo : public Simple {
public:
    Punto getCentro() const;
    void setCentro(const Punto& centro);
    float getRadio() const;
    void setRadio(float radio);

    static Circulo* crear(const Punto& centro, float radio);

    void trasladar(float x, float y);
    void rotar(float angulo);

    virtual ~Circulo();
private:
    Punto centro;
    float radio;
    Circulo(const Punto&, float radio);
};
```

/ Circulo.cpp ***/***

```
Circulo::Circulo(const Punto& centro, float radio) {
    this->centro = centro;
    this->radio = radio;
}

Punto Circulo::getCentro() const {
    return centro;
}

void Circulo::setCentro(const Punto& centro){
    this->centro = centro;
}

float Circulo::getRadio() const {
    return radio;
}

void Circulo::setRadio(float radio) {
    this->radio = radio;
}

Circulo* Circulo::crear(const Punto& centro, float radio) {
    if (radio <= 0) {
        throw std::invalid_argument("El radio debe ser mayor a 0.");
```

```
    }
    return new Circulo(centro, radio);
}

void Circulo::trasladar(float x, float y) {
    this->centro = this->centro.trasladar(x,y);
    Figura::trasladar(x, y);
}

void Circulo::rotar(float angulo) {
    this->centro = this->centro.rotar(angulo);
    Figura::rotar(angulo);
}

Circulo::~Circulo() {
}
```